Programa Concorrente para Integrar Numericamente Funções Elementares

Ricardo Kaê - DRE 116 039 521

Esse trabalho aborda aspectos de concorrência de um programa de integração numérica de funções elementares. Refere-se por **funções elementares**, funções de uma única natureza: somente polinomial ou somente trigonométrica etc. Não sendo a composição e nem a combinação de outras. Por exemplo, o programa não integra (automaticamente) a composição $f(x) = e^{sin(\sqrt{1-x})}$ ou a combinação linear $f(x) = 2sin(x) + 3cos(x)$ ou ainda a combinação não linear $f(x) = x^2sin(x) + 4x^3cos(x)$. Contudo, o programa integra numericamente polinômios de grau n, funções trigonométricas no geral (senoides, cosenoides, arcosenoides, tangente etc) e funções de natureza exponencial (exponencial e logaritmo natural na base e). Desde que elas aparecem sozinhas no integrando, sem estarem compostas ou combinadas com outras funções.

A integração dessas funções é feita automaticamente, isto é, com o usuário apenas passando as informações dos parâmetros de cada função, não precisando programá-las diretamente em C. Contudo, para funções específicas, como $f(x) = e^{-x^2}$ ou $f(x) = \sqrt{4 - x^2}$, o usuário precisa programá-las se quiser efetuar a integração numérica pelos métodos desse programa. Na realidade, para essas duas funções citadas, foi colocado um módulo extra ao programa que permite integrá-las automaticamente, como as funções elementares. Integrais essas que serviram de exemplo ao longo desse texto. Contudo, isso não acontece de maneira geral, de forma que para outras combinações ou composições de funções, cabe ao usuário final extender essas funcionalidades ao programa.

O uso do programa é baseado em 7 parâmetros, cujo o qual 2 são números mágicos para controlar as funcionalidades internas que se deseja.

```
Uso: ./integralc <MI> <N> <FE> <a> <b> <h> <nome_arquivo_parametros_funcao>
```

<MI> é o primeiro parâmetro, que indica o **modo de integração**. O programa integra em três modos:

1. Somas de Riemann (inserindo retângulos à esquerda, à direita e nos pontos do meio de uma região limitada)

2. Método Trapezoidal (inserindo trapézios na região limitada, que são polinômios interpoladores de grau 1 nos subintervalos)

3. A regra de Simpson (que é uma variação do modo trapezoidal para um polinômio interpolador quadrático nos subintervalos)

<N> é segundo parâmetro, que indica o **número de threads** a serem usadas na integração concorrente das funções.

<FE> é o terceiro parâmetro, que é o número mágico indicador da **função elementar** a ser integrada

```
// Números mágicos de cada função elementar
#define POLINOMIO 1
#define SENO 2
#define COSSENO 3
#define TANGENTE 4
#define SECANTE 5
#define COSSECANTE 6
#define COTANGENTE 7
#define ARCOSENO 8
#define ARCOCOSSENO 9
#define ARCOTANGENTE 10
#define ARCOSECANTE 11
#define ARCOCOSSECANTE 12
#define ARCOCOTANGENTE 13
#define EXPONENCIAL 14
#define LN 15
// Números mágicos das funções extra - começando a partir do 20
#define PI 20
#define EXPNX2 21
#define CNLTANEXP 22
```

<a, b, h> São os parâmetros que indicam: o extremo da **esquerda** a, da **direita** b do intervalo $[a, b]$ e o **passo** h (stepsize) de integração. Desse jeito, para o intervalo $[0, 1]$ em que deseja-se $n = 50$ particionamentos, calcula-se o passo da seguinte forma: $h = (b - a)/n = 1/50 = 0.02$

E o último parâmetro é um arquivo ASCII com uma única linha que recebe números, que são os coeficientes (ou **parâmetros** de cada função <FE> selecionada). A formatação do arquivo é quase a mesma para todas as funções, com exceção dos polinomios e das funções extras. Para as funções extras pode-se passar qualquer arquivo de configuração, até mesmo um vazio. Só não pode deixar de passar um arquivo, pois o programa espera receber por um, para executar.

Formatação do arquivo ASCII
Para polinômios, a configuração da linha do arquivo é a seguinte:

$$< N >< A >< B >< C > ... < N >$$

onde N é o primeiro número indicativo do grau do polinômio e os números A,B,C,...,N são os coeficientes do polinômio de grau n. Desse jeito, para o polinômio $p(x) = 5x^2 + 4x$, têm-se a seguinte linha para um arquivo de configuração:

2 0 4 5

Onde 0 é o coeficiente do termo constante, 4 o coeficiente do termo linear e 5, o coeficiente do termo quadrático. E 2, o primeiro parâmetro é o indicador do grau do polinômio.
Para o polinômio de grau 12, $p(x) = 5x^{12} + 7x^5 + 150$, a linha do arquivo se mostra na seguinte forma:

12 150 0 0 0 0 7 0 0 0 0 0 0 5

12, o grau. 150, o termo constante, 7 o coeficiente do termo de quinta ordem e 5, o coeficiente do termo de décima segunda ordem de $p(x)$

Para o restante das funções computadas pelo programa, isto é, as funções exponenciais, logarítmicas e trigonométricas, a configuração do arquivo é a mesma (de forma que pode se reaproveitar os arquivos de configuração de uma função para a outra, apenas trocando o <FE>). A linha desses arquivos são exclusivamente caracterizada por **três** números:

um número C, um número ω e um número k.

E o calculo da função trigonométrica (ou exponencial) escolhida é dado da seguinte forma. Digamos que seja selecionado o SENO em <FE>, então o valor computado será: $Csin(\omega x + k)$. Se for selecionado TANGENTE em <FE>, então o valor computado será: $Ctan(\omega x + k)$. Se for selecionado LN em <FE>, então o valor computado será $Clog(\omega x + k)$. Se for selecionado EXPONENCIAL em <FE>, então o valor computado será $Cexp(\omega x + k)$ e assim para todas as funções desse programa, com exceção dos polinômios e das funções extras.

Logo, se deseja-se integrar a função $f(x) = 5sin(2x)$ no intervalo $[0, \pi/2]$ pelo programa. Deve-se fazer um arquivo ASCII com a seguinte linha 5 2 0. Chamemos esse arquivo de *sin* para esse exemplo. E em seguida chamar o binário do programa da seguinte forma:

```
$ ./integralc 1 4 2 0 1.570796 0.02 sin
f(x) = (5.00) * sin(2.00 * x + 0.00)
        onde C = 5.00, w = 2.00, phi = 0.00
# Método de Somas de Riemann Sequencial. Particionado em retângulos...
Ln: 4.997671
```

```
Rn: 4.999830
Mn: 4.999751
Número de partições 78
Resultado da integral Sequencial: 4.9997505293180
# Método de Somas de Riemann Concorrente. Particionando em retângulos...
Número de partições 78
Resultado da integral Concorrente: 4.9997505293180
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
78              0.000113        4               0.000661        0.171385
```

Note o 1 como o primeiro parâmetro na chamada do programa, indicando o modo de integração por somas de Riemann. O número de threads vindo na sequência. O número mágico <FE> = 2, para indicar o SENO. O intervalo de 0 a $\pi/2 \approx 1.570796$, o passo $h = 0.02$ que ofereceu 78 partições e o arquivo *sin* contendo as configurações para a função SENO escolhida em <FE>. O resultado é aproxidamente 5, que é o valor dessa integral definida no intervalo considerado.

Como outro exemplo da execução do programa. Considere a função $f(x) = 10x^2 + 25x$ no intervalo de [0,2]. A primitiva dessa função é $F(x) = 10x^3/3 + 25x^2/2$. Logo, o valor da área é dado por:

$$F(2) - \cancel{F(0)} = 80/3 + 50 = 230/3 = 76.66666666666667$$

Desse jeito, para integrar essa função no programa, deve-se criar um arquivo, digamos para esse exemplo que o nome seja $x2$ com a seguinte linha : 2 0 25 10. E chamar o programa da seguinte forma

```
$ ./integralc 2 8 1 0 2 0.000001 x2
f(x) = (10x^2) + (25x^1) + 0
# Método trapezoidal Sequencial. Particionando em trapézios...
Número de partições 1999999
Resultado da integral Sequencial: 76.6665766667350
# Método trapezoidal Concorrente. Particionando em trapézios...
Número de partições 1999999
Resultado da integral Concorrente: 76.6665766667350
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
1999999         0.129297        8               0.033320        3.880449
```

Nota-se nesse exemplo que o 2 é o indicativo do modo de integração trapezoidal, 8 foram o número de threads utilizadas, <FE> = 1, o indicador do POLINOMIO e o intervalo foi $[0, 2]$ com passo bem pequeno, $h = 0.000001$, dividindo tal intervalo em 1999999 partições. E assim, a versão concorrente do método trapezoidal, nesse caso, apresentou ganhos em relação ao método trapezoidal sequencial. Já que o número de particionamentos aumentou consideravalmente.

O último exemplo a ser feito calculará a seguinte integral $\int_0^2 5e^{2x}\ dx$ pela regra de Simpson, que é o terceiro modo de integração desse programa. Note que os coeficientes $C = 5$ e $\omega = 2$ são iguais aos usados na integração de $f(x) = 5sin(2x)$, logo o arquivo *sin* pode ser reutilizado para essa integral, apenas trocando o <FE> de 2 (indicador do SENO) para 14 (indicador da EXPONENCIAL). Desse jeito, a chamada do programa se mostra na seguinte forma

```
$ ./integralc 3 8 14 0 2 0.00000002 sin
# Programa Concorrente para integrar funções elementares
f(x) = (5.00)e^(2.00 * x)
Número de partições 100000000
Resultado da integral Sequencial: 133.9953750828606
# Concurrent Simpson's rule.
Número de partições 100000000
Resultado da integral Concorrente: 133.9953750828606
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
100000000       4.309709        8               0.695361        6.197797
```

Usando um outro sistema de computação algébrica, pode-se comparar os resultados obtidos. Nesse texto foram usados o Máxima, um poderoso sistema de computação simbólica e o Geogebra.

O Máxima retorna como resultado para essa integral

$$5\int_0^2 e^{2x}\ dx = 5\left(\frac{e^4}{2} - \frac{1}{2}\right) = 133.9953750828606$$

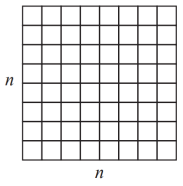
que é o mesmo resultado encontrado pelo programa. Contudo, vale mencionar que o Máxima não usa algoritmos de integração numérica. Ele encontra de fato a primitiva do integrando e aplica nos extremos considerados.

Nas próximas seções desse trabalho aborda-se a necessidade da integração numérica e da concorrência como uma técnica de auxílio para esses métodos numéricos de integração, de forma a trazer mais acurácia e eficiencia nos resultados, bem como os casos de teste e análise de desempenho em que foram submetidos o programa apresentado.

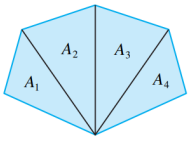
1 Integração Numérica

De maneira geral, a integral resolve o problema de determinação de áreas de regiões quaisquer limitadas num intervalo considerado. Tal problema foi estudado inicialmente pelos Gregos nas pré civilizações e se repercutiu até meados do século XVI - XVII, quando foi resolvido formalmente por Newton e Leibniz com o **Cálculo**, que levou aos conceitos de Integral, Limites, Derivadas, o Teorema Fundamental do Cálculo, Séries Infinitas, Equações Diferenciais, Transformadas Integrais e muitos outros. Sendo o maior crédito do Cálculo nesse contexto geométrico, ao Lebniz devido à sua preocupação com a geometria, enquanto que o Newton tinha um olhar mais físico nessa época, como citam as literaturas. Vale mencionar também, a contribuição do Leibniz para as notações no Cálculo: os símbolos de derivação $\frac{dy}{dx}$, $\frac{dy}{dt}$, como razões de taxas diferenciais e o símbolo de integração $\int$, que lembra um S alogando, foram denotados por ele e esse último expressa bem, a ideia básica no cálculo da integral, como uma soma ou um limite de somas.

A ideia da integral como um somatório de parcelas infinitesimais é um refinamento do método de Exaustão encontrado nos trabalhos de Euclides e Eudoxo, que consiste no processo de determinar a área de regiões quaisquer limitadas, inserindo polígonos nessa região, em que sabe-se as áreas de cada um. Dito de outra forma, o método consiste em particionar a área de uma região limitada em polígonos cujo se conhece a área de cada um. E portanto, a área da região limitada fica determinada pela soma das área de cada polígono inserido (ou de cada partição efetuada). A medida em que aumenta-se o número de partições, aproxima-se melhor a área da região pela soma das áreas dos polígonos regulares.

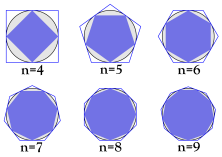


(a) Área de um quadrado de lado n é a soma das áreas de um quadrado de lado 1



$A = A_1 + A_2 + A_3 + A_4$

(b) Insere-se polígonos com áreas A_1, A_2, A_3, A_4 e calcula-se A. Ou de outra forma, particiona-se A em regiões de área A_1, A_2, A_3, A_4



(c) Quadratura grega do círculo com inscrição e a circuninscrição de polígonos regulares cada vez maiores. Processo feito por Arquimedes para limitar inferiormente e superiormente a área do círculo

Figura 1: Método da Exaustão

Esse método foi explorado por Arquimedes (Figura 1c), para calcular uma aproximação para π, preenchendo um círculo internamente e externamente com

polígonos regulares (de lados) cada vez maiores, permitindo assim delimitar cada vez mais, a área do círculo inferiormente e superiormente. E com a área do círculo é possível se calcular o valor de π (ou pelo menos estimar).

A integral consiste de um refinamento desse mesmo processo: particionar uma região definida por uma função $f(x)$ contínua e limitada num intervalo $[a, b]$ em subregiões e calcular a área dessas subregiões. A área total da região definida e limitada por f nesse intervalo é dada pela soma das áreas de cada particionamento. A medida que aumenta-se o número de subintervalos (particionamentos), obtêm-se uma aproximação cada vez maior para a área total. Essa ideia é definida de maneira moderna no Cálculo como somas de Riemann, o que leva a uma primeira forma de cálcular a integral definida em $[a, b]$, que é uma forma numérica.

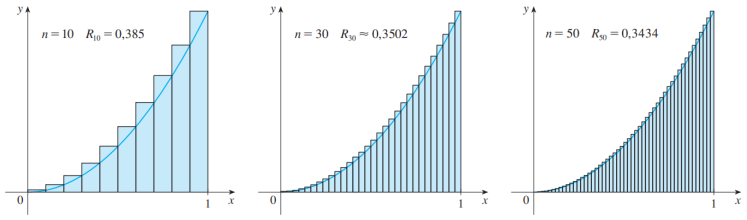


Figura 2: Somas de Riemann à direita R_n para calcular a área de x^2 em $[0, 1]$, onde n é o número de retângulos (particionamentos) efetuados

A segunda maneira de calcular a integral é feita indefinidamente, isto é, sem necessitar do intervalo (limites) de integração, aplicando o teorema fundamental do cálculo, que diz que a derivada e a integral são processos simétricos quando aplicados a uma função. De outra forma, diz que a integral é o processo de anti derivação de uma função f qualquer. Desse jeito, encontrar o resultado da integral de f consiste em encontrar funções primitivas F, que são funções cujas derivadas dão f, o valor do integrando, $F' = f$. O que leva a uma série de regras analíticas (substituição simples, integração por partes, frações parciais etc) para se anti derivar e encontrar primitivas de funções, que é uma forma de manipulação simbólica.

Ambos os métodos funcionam para se calcular a integral e se obter a área de certas regiões, contudo dependendo do caso, há preferência da aplicação de um em relação a outro. Para a solução mecânica da integral, isto é, manual, emprega-se o segundo, por fins de praticidade. É mais facil manipular simbolicamente o integrando segundo um conjunto de regras bem definidas do que realizar um número muito grande (infinito) de somas e trabalhar com números com muitas casas decimais.

No entanto, para computadores, isso não é um problema, já que eles lidam bem com com números muitos grandes e com tarefas repetitivas com precisão,

de forma que não há razão para não se usar primeiro método em computadores.

Um outro ponto quanto a preferência do primeiro método em relação ao segundo, está na facilidade de se programar. A programação do primeiro método é mais fácil do que o segundo, visto que os algoritmos numéricos se encaixam bem com os paradigmas procedurais (imperativos e orientados à objetos), que são os padrões da maioria das linguagens de programação atuais. Enquanto que os métodos simbólicos se encaixam melhor com linguagens declarativas ou funcionais, que já não demandam de tanta disponibilidade quanto as procedurais. Basta ver a quantidade de sistemas de computação numérico em relação a quantidade de sistemas de computação simbólico.

Sistemas Numéricos	**Sistemas Simbólicos**
Octave, MATLAB, Mapple, Mathematica, Geogebra	Axiom (antigo Scratchpad), Máxima

Desses sistemas bem conhecidos, pode-se notar que a quantidade numérica é o dobro dos simbólicos, o que indica a disponibilidade de um em relação a outro.

Uma outra razão pela preferência por métodos numéricos de integração em relação aos simbólicos é que nem sempre é facil ou mesmo possível se encontrar primitivas para uma função f qualquer. Para as funções elementares e suas combinações lineares, essa tarefa é facil, pois os valores da integral já encontram-se tabulados (Figura 3).

$$\int cf(x)\,dx = c\int f(x)\,dx \qquad \int [f(x)+g(x)]\,dx = \int f(x)\,dx + \int g(x)\,dx$$

$$\int k\,dx = kx + C$$

$$\int x^n\,dx = \frac{x^{n+1}}{n+1} + C \quad (n \neq -1) \qquad \int \frac{1}{x}\,dx = \ln|x| + C$$

$$\int e^x\,dx = e^x + C \qquad \int a^x\,dx = \frac{a^x}{\ln a} + C$$

$$\int \operatorname{sen} x\,dx = -\cos x + C \qquad \int \cos x\,dx = \operatorname{sen} x + C$$

$$\int \sec^2 x\,dx = \operatorname{tg} x + C \qquad \int \operatorname{cossec}^2 x\,dx = -\operatorname{cotg} x + C$$

$$\int \sec x \operatorname{tg} x\,dx = \sec x + C \qquad \int \operatorname{cossec} x \operatorname{cotg} x\,dx = -\operatorname{cossec} x + C$$

$$\int \frac{1}{x^2+1}\,dx = \operatorname{tg}^{-1} x + C \qquad \int \frac{1}{\sqrt{1-x^2}}\,dx = \operatorname{sen}^{-1} x + C$$

$$\int \operatorname{senh} x\,dx = \cosh x + C \qquad \int \cosh x\,dx = \operatorname{senh} x + C$$

Figura 3: Tabela de Integrais indefinidas

Contudo, para funções da forma g ou h,

$$\mathbf{g}(x) = e^{-x^2} \implies \int_0^2 e^{-x^2}\,dx \qquad \mathbf{h}(x) = 5x^3e^x + 2x^2tan(x) \implies \int_0^1 5x^3e^x + 2x^2tan(x)\,dx$$

que são composições ou combinações não lineares dessas funções elementares, tais métodos simbólicos já não costumam serem tão fáceis (ou eficientes) de serem aplicados, seja manualmente ou computacionalmente, enquanto que o

método numérico calcula com eficácia o valor dessas funções num conjunto de pontos e consegue integrar. O programa apresentado nesse trabalho calcula de maneira geral a integral somente para funções elementares, de forma que as funções g, h apresentadas acima não fazem parte de seu domínio. Contudo, acrescentando-se um módulo extra ao programa consegue-se integrar exclusivamente essas funções e obtêm-se os resultados mostrados na Figura 4.

Vale mencionar que o resultado da integral de $h(x)$ em [0,1], calculado pelo Máxima, um poderoso sofware de computação simbólica é dado por

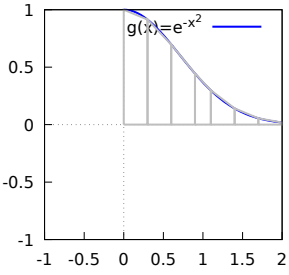
$$\int_0^1 5x^3e^x + 2x^2tan(x)\ dx =$$
$$-\frac{\left(24i\ atan\left(\frac{sin(2)}{cos(2)+1}\right) + 12ln(2cos(2)+2) + 12li_3(-e^{2i}) - 24i\ li_2(-e^{2i}) + 9zeta(3) - 8i + 120e - 360\right)}{12}$$

o que é claramente um resultado não trivial de interpretar.

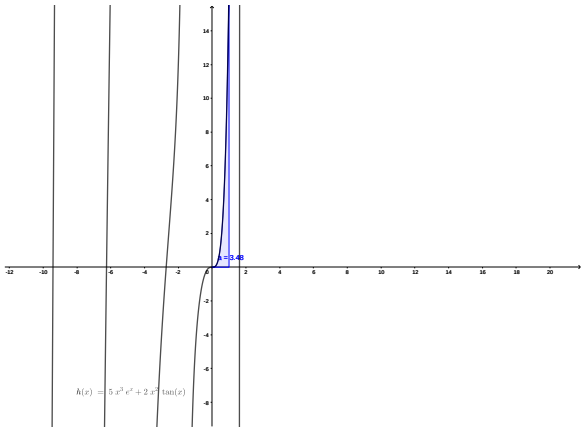
```
$ ./integralc 2 4 21 0 2 0.0005 x2
f(x) = e^(-x^2)
# Método trapezoidal Sequencial. Particionando em trapézios...
Número de partições 40000
Resultado da integral Sequencial: 0.8820813907472
# Método trapezoidal Concorrente. Particionando em trapézios...
Número de partições 40000
Resultado da integral Concorrente: 0.8820813907472
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
40000           0.006377        4               0.003210        1.986836

$ ./integralc 3 4 22 0 1 0.00005 x2
f(x) = 5x^3e^x + 2x^2tan(x)
Número de partições 20000
Resultado da integral Sequencial: 3.4780353484434
# Concurrent Simpson's rule.
Número de partições 20000
Resultado da integral Concorrente: 3.4780353484434
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
20000           0.008535        4               0.003154        2.706346
```

Figura 4: Calculo das integrais das funções g, h apresentadas acima pelo programa *integralc*



(a) Trapézios inseridos em $g(x) = e^{-x^2}$. A área com extrema precisão dá 0.8820813907624215 - calculado pelo Máxima aqui.



(b) A integral calculada pelo Geogebra da função $h(x) = 5x^3e^x + 2x^2tan(x)$ em [0,1]

Figura 5: Plot das funções g, h

Desse jeito, percebe-se a importância e necessidade dos métodos numéricos de integração. E tais métodos podem se beneficiar da concorrência e do paralelismo dos computadores atuais para calcular um resultado mais preciso para integral, com maior número de particionamentos de maneira mais rápida.

É claro que é observado que com centenas ou milhares de particionamentos (da ordem de 10^3, 10^4 etc), o resultado da integral já têm uma acúracia muito grande, já que os polígonos inseridos preenchem quase que completamente a região limitada e esses já são efetuados de maneira muito rápida. Por exemplo, para 40.000 partições feitas acima, o tempo de execução sequencial é de 0.006377 s (ou seja, é instântaneo) e a acurácia do resultado já é muito grande também.

Se usarmos o Máxima como uma métrica de comparação, percebe-se que o programa (Figura 4) chegou bem próximo do resultado da integral dessa função $g(x) = e^{-x^2}$ em [0,2] calculada por ele (Figura 5a).

Logo, pode-se questionar se a concorrência é de fato necessária nesses casos. Ou seja, se são necessário tantos particionamentos assim, para a concorrência valer a pena. E a resposta para essa questão é que sim, é necessária. Seja somente pelo fato de ser um método mais eficiente de extrair processamento dos computadores modernos, como também pelo fato de se obter cada vez mais acurácia nos resultados já obtidos. Por exemplo, em certas aplicações, obter mais acúracia implica em uma tecnologia mais avançada. Pode-se citar como exemplo a manufaturação computadorizada de circuitos integrados, que usa de um computador para particionar a área de regiões de uma placa de circuito,

para inserir chips lá dentro. Quanto o maior o número de particionamentos, melhor pode ser feito esse cálculo de quantos chips podem ser inseridos e portanto melhor é a tecnologia produzida. E para particionamentos da ordem de $10^8, 10^9$, que implicam em centenas de milhares ou bilhares de chips, uma aplicação sequencial já não é tão rápida assim, de maneira a trazer precisão nos resultados.

De maneira a tornar prático, o que vêm sendo dito, da necessidade da concorrência, é que mostro os resultados da integral abaixo

$$\int_0^2 \sqrt{4 - x^2} = \pi$$

calculado pelo programa desse trabalho. A integral considerada é igual a π no intervalo de [0,2]. De forma que com mais particionamentos, tem-se uma aproximação numérica cada vez mais correta para π, nesse caso.

Usando o Máxima como métrica, que retorna o seguinte valor para $\pi = 3.141592653589793$. Tem-se os resultados obtidos pelo programa, em que são marcados os digitos com erro quando comparados com valor do Máxima.

```
$ ./integralc 2 8 20 0 2 0.001 x2
f(x) = sqrt(4 - x^2)
Resultado da integral Sequencial: 3.1415(795059120)
Resultado da integral Concorrente: 3.1415(795059120)
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
2000            0.000330        8               0.002203        0.149885

$ ./integralc 2 4 20 0 2 0.00001 x2
f(x) = sqrt(4 - x^2)
Resultado da integral Sequencial: 3.1415926(404419)
Resultado da integral Concorrente: 3.1415926(404419)
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
200000          0.015855        8               0.004483        3.536988

$ ./integralc 2 4 20 0 2 0.000001 x2
f(x) = sqrt(4 - x^2)
Resultado da integral Sequencial: 3.14159265(11740)
Resultado da integral Concorrente: 3.14159265(11740)
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
1999999         0.091581        8               0.022165        4.131807

$ ./integralc 2 4 20 0 2 0.0000001 x2
f(x) = sqrt(4 - x^2)
```

```
Resultado da integral Sequencial: 3.1415926535(766)
Resultado da integral Concorrente: 3.1415926535(766)
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
20000000        0.881927        8               0.150227        5.870618

$ ./integralc 2 4 20 0 2 0.00000001 x2
f(x) = sqrt(4 - x^2)
Resultado da integral Sequencial: 3.141592653589(4)
Resultado da integral Concorrente: 3.141592653589(4)
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
200000000       8.410704        8               1.427894        5.890286

$ ./integralc 2 8 20 0 2 0.000000001 x2
Número de partições 2000000000
Resultado da integral Concorrente: 3.1415926535897(8)
# LOG TEMPO #
PARTIÇÕES       Tseq            THREADS         Tconc           GANHO
2000000000      0.000000        8               16283.911909    0.000000
```

Para essa última execução, nem se efetuou o cálculo da integral sequencial devido a demora no processamento. Como os tempos estão em segundos. Essa última execução demorou em torno de 4h - 4h:30 de processamento e obteve via integração uma aproximação para π exatamente como a do Máxima, com um erro de apenas uma casa decimal. Sendo que o Máxima não calcula o π via integração, visto que, sem dúvidas esse é um processo bem ineficiente para fazê-lo. Mas com esse exemplo, torna claro a necessidade de se ter mais particionamentos em uma determinada integral visando mais acurácia e ainda, se os ganhos forem observados, torna-se claro como a concorrência pode ajudar, acelerando as aplicações nesses casos de muitos particionamentos.

Na Economia ou em Jogos eletrônicos é certo de se terem aplicações que demandam de integrais altamente acuradas. A concorrência nesses casos pode ajudar.

2 Projeto do programa e das soluções concorrentes

O projeto do programa foi implementado em 4 diferentes módulos:

1. Um módulo *principal.c*, que contem a rotina principal (main) do programa, que chama todas as outras
2. Um módulo *integral.c*, que contem as implementações dos métodos de

integração sequencial e concorrentes do programa, que são declaradas em *integral.h* (seu header). Bem como as estruturas de dados e funções auxiliares a esses processos de integração, como a implementação da *barreira* de threads, usadas em alguns dos modos de integração concorrente. Ou a função que seleciona o modo de integração escolhido pelo usuário. Todas encontram-se nesse módulo.

3. Um módulo $felementar.c$, que contem as implementações das funções elementares integradas pelo programa.

4. E por fim, um ultimo módulo $fextras.c$ que contem as implementações das funções extras a serem integradas pelo programa

   Esses dois últimos módulos também contem arquivos de cabeçalhos, $felementar.h$ e $fextras.h$ contendo as declarações das funções usadas nesse contexto.

A ideia principal era construir um programa concorrente para integração numérica sem o uso de menus, o que levou a estratégia de usar números mágicos, que é algo fora de moda atualmente, até por extender a documentação dos programas, mas que se encaixou bem para efetuar testes desejados, já que não há interação com usuário ou recompilação.

Para as soluções concorrentes, a ideia geral foi dividir o cálculo das áreas entre as threads, de forma que cada thread tem a sua soma local, que é a soma de um certo (sub)conjunto de áreas e criar uma variável global para armazenar a soma da contribuição de cada soma local de cada thread. Desse jeito, há uma condição de corrida no programa, que foi resolvida usando *locks*, visto que esse é o método mais simples de sincronização de threads. Assim as threads se excluem mutuamente para escrever o compito de suas somas locais na variável global e a thread só escreve se ela é possuidora do *lock*. O programa só encerra assim que todas as threads terminaram de escrever suas contribuições na variável global.

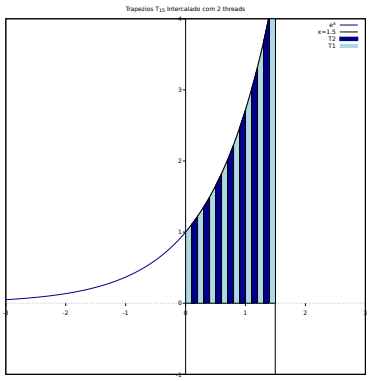
Nos métodos trapezoidal e de Simpson fez-se necessário o uso de *barreiras*, para sincronizar coletivamente as threads a partir de um certo ponto. Essa decisão acabou atrasando um pouco a aplicação frente ao método de Riemann, que não usa desse artifício, mas que ainda assim não diminuiu os ganhos obtidos pela aplicação. Como é mostrado na análise de desempenho, foram obtidos ganhos de 4 - 6 no programa quando executados com 8 threads, que é o número máximo de threads que esse computador consegue lidar.

Estratégias de solução concorrente para integração numérica
De maneira geral, o problema de integração numérica pode ser visto naturalmente como um problema concorrente, visto que, o problema por si só, já envolve particionamento de um problema maior: O cálculo da área de uma região qualquer limitada a partir de problemas menores e que podem ser resolvidos independentemente: que são os cálculos das área de cada particionamento efetuado (subintervalo). Assim, de maneira simplista, basta deixar que cada thread

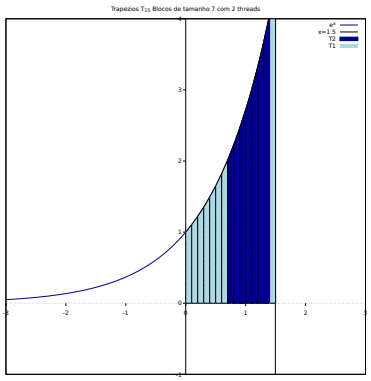
faça o compito de cada particionamento e sincronizar as threads para entregar o resultado final corretamente.

Contudo como o o processador só consegue lidar com um conjunto limitado de threads simultaneamente, que faz sentido dividir as tarefas entre elas em um número menor ou igual a esse que o processador consegue lidar. Do contrário, se a computação for feita com mais threads do que o computador consegue lidar, digamos uma para computar cada particionamento, só haverá atrasos na aplicação, visto que, ele só vai lidar com um conjunto limitado simultaneamente e as demais ficarão aguardando em filas de espera, até serem escalonadas para executar.

Dito isso, há duas abordagens para dividir os particionamentos entre as threads, de forma que cada thread efetue um subconjunto de somas locais: A abordagem intercalada e a abordagem em blocos contíguos. Para apresentar tais abordagens, considere a região abaixo definida pela função $f(x) = e^x$ no intervalo de $[0,\ 3/2]$, que será dividida em 15 partições de tamanho $h = 0.1$, onde a integração será feita por 2 threads.



(a) Trapézios inseridos em $f(x) = e^x$ no intervalo $[0, 3/2]$. Duas threads computaram a soma de maneira intercalada.



(b) Trapézios inseridos em $f(x) = e^x$ no intervalo $[0, 3/2]$. Duas threads computaram a soma em blocos contíguos de tamanho 7

Figura 6: Abordagens de divisão de tarefas entre as threads

Na figura 6*a*, observa-se que as partições são divididas entre as threads de maneira intercalada, assim elas vão computando as somas se intercalando umas com as outras, até preencherem a região limitada. E depois escrevem seus resultados na soma global. E na figura 6*b*, observa-se que a região limitada que foi dividida em 15 partições foi subdividida em dois blocos contíguos entre as 2 threads, de tamanho 7 cada um. A primeira thread computa 7 somas. Depois a segunda thread computa 7 somas. E então volta para a primeira, até preencherem completamente a região, como é mostrado na figura. Em ambos as

estratégias, nota-se o balanceamento de cargas entre as threads.

Como pode ser visto, ambas as abordagens dariam certo para computar a integral. Mas a escolhida para implementação das funções concorrentes do programa foi a primeira, a de intercalação entre as threads. E não houve uma razão específica para tal, foi apenas a decisão tomada.

3 Casos de Teste

Os casos de teste elaborados para o programa foram pensados para equalizar os testes em dois sentidos:

1. Os parametros de cada modo de integração.

   Todos os testes foram realizados na mesma função $f(x) = cos(x)$, sendo a integral computada a seguinte:

   $$\int_{-3}^{3} cos(x)\ dx$$

   Isso equaliza os parâmetros de cada modo de integração, já que todos recebem a mesma função, no mesmo intervalo e computam a mesma integral, por métodos diferentes. O que permite avaliar o desempenho de um método de integração em relação a outro.

2. Os ganhos do programa em relação ao número de threads usado.

   Na presente sessão já se sabe que o problema de integração concorrente só se torna CPU bounded (desafiador para CPU), quando o número de particionamentos cresce consideravelmente. Sendo assim, com o número de particionamentos grande, obtêm-se maiores ganhos, com o maior número de threads que o processador consegue lidar. Contudo, se o número de particionamentos é baixo, usar o maior número de threads nem sempre é uma estratégia favorável. Pode-se conseguir ganhos maiores com número menor de threads, nesse caso, que é o que será mostrado a seguir.

   A escolha do número de threads depende mutuamente do número de particionamentos realizado. Uma escolha menor de threads para um número menor de particionamentos pode se mostrar melhor do que para um número maior de threads (para esse mesmo particionamento), visto que, o custo para criar e escalonar muitas threads pode ser maior do que o tempo de execução da função concorrente para um número pequeno de threads.

   Dito isso, foram pensados em 4 ordens de grandeza para os particionamentos, visando equalizar os testes nesse segundo sentido, quanto aos ganhos em relação a cada thread. Os particionamentos foram 10^3, 10^6,

10^7 e 10^8. Foi excluído a ordem de bilhares (10^9) de particionamentos, visto que a presente máquina se mostrou muito lenta para executar tais casos (como foi citado na aproximação para π). Então, a maior ordem considerada foi 10^8.

Foram 3 casos de testes realizados. Um para cada modo de execução, na integral dita acima, na função cosseno avaliada de -3 a 3.

$$f(x) = \int_{-3}^{3} cos(x)\ dx$$

Em cada caso, computou-se a integral para os particionamentos listados: 10^3, 10^6, 10^7 e 10^8. Em cada particionamento, executou-se o programa 5 vezes para os números de threads listados: 1, 2, 4 e 8. Totalizando 20 execuções do programa por particionamento, 5 para cada thread e 80 execuções do programa para cada caso. Como foram 3 casos de execução, foram ao todo 240 execuções do programa. Essas execuções foram automatizadas por um script *tabela.sh*, que encontra-se junto com o código fonte do programa desse trabalho. A tabela gerada permitiu gerar 3 gráficos de desempenho, um para cada modo de execução, que serão apresentados na sequência.

4 Avaliação de Desempenho

A partir dos testes realizados foi possível gerar os gráficos de ganho por threads, para cada particionamento. Foram 3 gráficos no total, um para cada modo de execução, como pode ser visto abaixo.

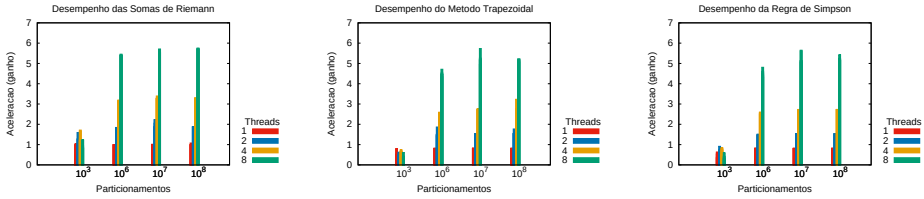


Figura 7: Gráficos de ganhos da aplicação por threads para cada particionamento

Os gráficos foram obtidos a partir de conjuntos de dados retirados da tabela gerada pelo script *tabela.sh*. Essas tabelas também se encontrarão junto com o código fonte do programa, para verificação das informações.

Dos gráficos torna-se importante mencionar duas coisas:

1. Para o particionamento de 10^3, observa-se o que foi dito na sessão anterior. O programa com 2 ou 4 threads, se comporta melhor do que com 8 threads. Ele atinge mais ganhos com 2 ou 4 threads do que com 8, visto

que o custo para criar e escalonar 8 threads é maior do que o tempo de execução do programa para um número menor de threads (2 ou 4). Isso para todos os modos de integração. Logo, a escolha do número de threads para integração tem que ser baseada no número de particionamentos para atingir melhor perfomance.

2. A Soma de Riemann apresentou melhores resultados do que os outros métodos. De forma, que pode-se dizer analisando somente os gráficos que ele é o melhor método de integração. (o que oferece maior ganho à aplicação). Contudo, apesar da aplicação ter tido mais performance com Riemann, ele é o método mais impreciso dos três. E além disso, por motivos de implementação, os outros métodos usaram *barreiras* de threads, para sincronização coletiva, o que provalvelmente pesou nos resultados.

5 Considerações finais

O programa apresentado nesse trabalho não satisfaz a exigência que a maioria dos softwares de computação numérica possuem em seus pacotes de integração, que é o fato de se poder integrar (numericamente) funções não elementares gerais, que são bem mais complicadas de serem integradas pelos métodos simbólicos. Pra incorporar tais funcionalidades ao programa, seria necessário um *parser* (analisador sintático) para reconhecer as funções não elementares de entrada ao programa por parte do usuário. Entre outros conceitos da área de compiladores.

Contudo, depois dos argumentos apresentados e dos resultados obtidos pelo programa, mostrados nos gráficos e em tabelas, o presente trabalho tenta tornar visível como a concorrência é aliada dos algoritmos numéricos, de maneira geral, para se obter melhor perfomance, apresentando especialmente o caso de integração. De forma a ser um argumento à favor da inclusão da concorrência nesses pacotes algébricos de computação numérica. De maneira geral, a concorrência embutida nesses pacotes ajudaria não somente no problema de integração, como a muitas outras áreas a que esses pacotes se destinam, como Machine Learning, Computação Científica etc.

O *Octave*, que é um *open source CAS – Computer Algebra System* escrito em C++, possui diversos módulos de integração. Alguns dos quais não foram incluídos nesse trabalho, como a Quadratura Gaussiana em diversas regras (Lobatto rules ou Gauss-Konrod rule) ou a Quadratura Clenshaw–Curtis, que é uma variação da Quadratura Gaussiana, também é suportada pelo programa, além de outras formas de integração. A própria documentação dessas funções disponível no site do *Octave* afirma que o melhor método de integração depende do integrando a ser usado. Contudo nem um dos métodos oferece a possibilidade de se usar *threads* para a computação, o que é certamente uma perda para o programa.

Enquanto que em outros sistemas, onde pode-se citar o *Julia*, por exemplo, uma linguagem *open source* projetada para atender os requisitos de computação de alto desempenho numérico e científico, já existe suporte a concorrência e paralelismo internamente, o que certamente contribui para o *slogan* da tecnologia: *Escreva como python e rode como C*.

Portanto, o presente trabalho traz de maneira introdutória a noção de como a concorrência e os algoritmos numéricos, de maneira geral, estão cada vez mais próximos, de forma que devem ser integrados na busca por mais desempenho. Os testes a que se submeteram o programa mostram isso e as outras tecnologias existentes também.

Hardware usado

```
$ neofetch
```

```
        _,met$$$$$gg.          ricardo@Ricardo
     ,g$$$$$$$$$$$$$$$P.       ---------------
   ,g$$P"     """Y$$.".        OS: Debian GNU/Linux 10 (buster) x86_64
  ,$$P'              `$$$.     Model: GA-78LMT-USB3 6.0
 ',$$P       ,ggs.     `$$b:   Kernel: 4.19.0-18-amd64
 `d$$'     ,$P"'   .    $$$    Uptime: 2 days, 9 hours, 33 minutes
  $$P      d$'     ,    $$P    Packages: 3054 (dpkg), 8 (snap)
  $$:      $$.   -    ,d$$'    Shell: bash 5.0.3
  $$;      Y$b._   _,d$P'      Resolution: 1680x1050
  Y$$.    `.`"Y$$$$P"'         DE: LXDE
  `$$b      "-.__              WM: Openbox
   `Y$$                        WM Theme: Natura
    `Y$$.                      Theme: Kiwi [GTK2/3]
      `$$b.                    Icons: RosaHumanity [GTK2/3]
        `Y$$b.                 Terminal: lxterminal
           `"Y$b._             Terminal Font: Monospace 10
               `"""            CPU: AMD FX-8300 (8) @ 3.300GHz
                               GPU: AMD ATI Radeon 3000
                               Memory: 5468MiB / 7456MiB
```

6 Referências

- Classnotes of C library for Numerical Integration - Einführung in die Programmierung für Physiker 13/14 Marc Wagner
- Notas de Aula de Cálculo Numérico Integração Numérica - Angela Gonçalves - DCC URFJ
- James Stewart Cálculo volume 1
- Methods of Numerical Integration - Second edition - Philip J.Davis and Philip Rabinowitz